エージェントシミュレーション向けブロック型ビジュアル言語の試作

○山梨裕矢　佐々木晃（法政大学）

Prototyping of Block-Based Visual Languages for Agent Based Simulation

* Y. Yamanashi and A. Sasaki (Hosei University)

概要－　本研究では，エージェントシミュレーション(ABS)向けのブロック型ビジュアル言語の試作を行った．ブロック型ビジュアル言語の代表例 Scratch は，小学生を対象としたプログラミング教育のために開発され，プログラミングの知識のないユーザーでもソフトウェアを組むことができる特長を持つ．本研究では，これを応用して，シミュレーション環境 NetLogo，社会シミュレーション向け環境 SOARS における言語のブロック化の試みを行った．本発表では，これらの試作について考察を行うとともに，社会シミュレーションシステムへの応用について議論する．

キーワード: エージェントシミュレーション，ブロック型言語，教育向けシミュレーション

1. はじめに

本研究では，エージェントシミュレーション(ABS)向けのブロック型ビジュアル言語の試作を行った．本研究の狙いは，(1)シミュレーション言語の学習コストの削減，(2)開発効率の向上，(3)シミュレーション内容および実行結果の解釈の促進，以上の3点である．ブロック型ビジュアル言語の代表例Scratch[1]は，小学生を対象としたプログラミング教育のために開発され，プログラミングの知識のないユーザーでもソフトウェアを組むことができる特長を持つ．本研究では，これを応用して，シミュレーション環境NetLogo[2]のサブセット言語に対応するブロック形ビジュアル言語を試作した．また，社会シミュレーション向け環境SOARS[3]における言語のブロック化の試みを行った．本発表では，これらの試作について考察を行うとともに，社会シミュレーションシステムへの応用について議論する．

2. 背景

複雑化する社会では，一般の個々人が本質的には専門的な知識を要する難しい選択を迫られる場面が増えている．そのような中で、教育を目的とした社会シミュレーションは，現実の問題に対する直観的な理解を促進する有力なツールであり，様々な層の人びとが理性的に判断するための助けとなると考えられる。

社会シミュレーションを簡潔に記述し実現するための手法の一つにエージェントシミュレーションがある．そのようなABSは，汎用言語のフレームワークとして提供されるツールを利用して実現できる．これにより表現力の高いシミュレーションを構築することが可能である．また，シミュレーションに特化した言語や環境を提供するシステムとしてNetLogo，SOARSが挙げられる．これらのシステムでは，汎用言語の知識を前提とせず，シミュレーションの構築が可能である．

本研究では，特にシミュレーションを記述するための言語について着目し，直観的にプログラミング可能なビジュアルブロックを用いてシミュレーションを表現することを試みる．ブロックを採用したシステムとして，MIT Media Labで開発されたScratchが挙げられる．これはもともと子供向けのプログラミング環境であるが，大学での初学者のプログラミング教育で用いられる事例など，様々な層の利用者を対象とした言語として着目されている．本研究では，エージェントシミュレーションの記述言語としてブロック言語を採用する試みを行っている．作成したプロトタイプでは，ブロック言語を開発するためのライブラリであるGoogle Blocklyを利用した．

3. NetLogo プログラムのブロックによる表現

今回の試みでは，まず，エージェントシミュレーションシステムの一つである NetLogo で利用可能なビジュアル言語を試作した．NetLogo の機能のサブセットに対応可能なビジュアルブロックを実装した．

NetLogo は LOGO 言語の派生として設計され，「タートル」「パッチ」「オブザーバー」という 3つの形式のエージェントを使ったプログラミングコンセプトとなっている．「タートル」はプログラム内で動くオブジェクトで「パッチ」は 2D 画面に格子状に配置されたものでタートルと違い動くことができない．「オブザーバー」はエージェントに命令を出し，動作を観測するものである．NetLogo ではこの３つの形式のエージェントが相互作用して動作する．

次に NetLogo の基本的な命令群とそれに対応するブロックの例をあげる．制御管理の命令には，ask, end, if, ifelse, repeat, set, to, to-report, while がある．タートルへの命令には，forward, create-turtles, distance, left, right がある．これに変数定義と変数への代入，新たなタートルを定義する命令文 breed などがある．以上に挙げた命令は今回のブロック型ビジュアル言語で実装した命令である．

プログラムの制御管理にあたる命令とそれに対応するブロックをみていく．*ask* は単一，もしくは複数のエージェントに命令を与える際に使用する関数である．NetLogo の構文では，ask *対象* [*処理内容*]となる．ask 命令をテキストで書く場合には対象エージェントと処理内容を記述しなければならないので，必須項目を記述するような形にすべきと考え，Fig.1 のようにし

た．

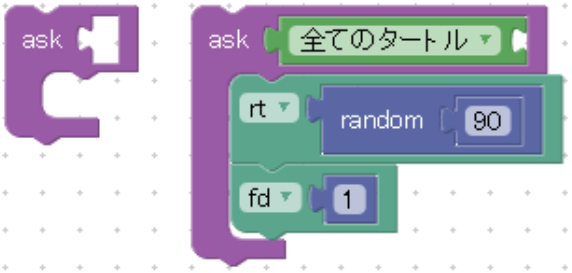

Fig. 1: ask ブロック

この ask ブロックではエージェント対象を与えなくてはならないので，ask ブロックにエージェントブロックを挿入するスペースを空けている．また，この ask ブロックで処理する内容を紐づけられるように他のブロックをこの ask ブロックの中に追加できるようにした．こうすることで，命令の引数に何を与えてやればよいか分かるようになる．

使用方法は次のようなものである．Fig.2 に記載した画面左部にあるカテゴリを押すと，そのカテゴリに属するブロックの一覧が表示される．ブロックの一覧から選択すると画面中央のプログラム編集部にそのブロックが出力される．それと同時に画面右のソースコード出力部にそのブロックに対応する NetLogo の関数が表示される．ブロックを組み合わせ，プログラムを作成していくとそれに合わせ画面右のソースコード出力部もリアルタイムにコードを出力する．

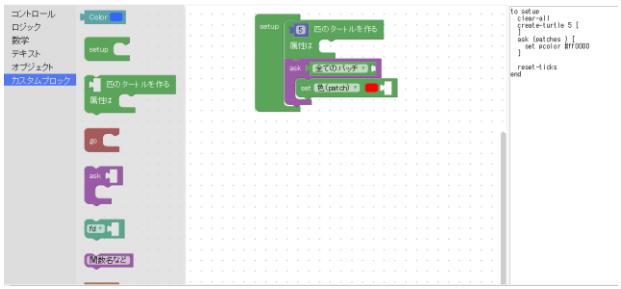

Fig. 2: プログラム全体図

次に，ブロック作成の方針について述べる．今回 NetLogo の各命令をブロック化する際にユーザーがその命令を分かりやすいように注意した点がある．NetLogo の関数の中には必須の引数と，必ずしも与えなくてよい引数がある．今回作成したブロックには，引数のブロックをブロックの内部に挿入する方法と，ブロックの外部に挿入する方法の二つがある．前者と後者を比較すると，前者はブロックの欠損度が強い．そのため，必須の引数はブロックの内部に挿入させ，オプションの引数はブロックの外部に挿入させるようにした．こうすることで，必須引数のブロックを必ず挿入させるように視覚的に表現した．また，Fig.3 のようにブロックの自己説明性を高めるためにブロックを折りたたむ時に意味の通じる文章になるようなテキストを挿入した．ブロック型ビジュアル言語の場合，テキストで書かれたプログラムに比べて幅をとるためブロックを折りたたむ必要性が出てくる．ブロックを折りたたんだ時にはブロックの形は単一の形になりテキストしか保持されないため，そこで意味の通る文章になるようにすべきである．このようにしておくことで，ブロックの自己説明性が高まることにもなる．

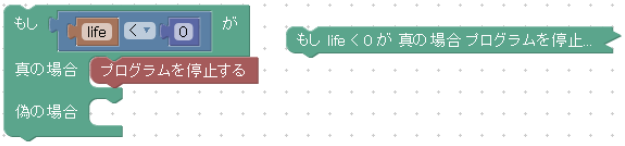

Fig. 3: 折りたたみ後のブロック

ブロックを作成する際には必ずそのブロックに型を設定した．こうすることで型チェックを行えるようになるためプログラムの保守性を高めることができる．今回のブロック型言語では，引数に期待する型と違う型のブロックは受け付けないようにすることで型チェックを実装した．例えばrandom関数では引数に必ず数字を受けとらなければならないので，文字列や真偽値を表すブロックの場合は引数にとれないようにした．

以下は，文献[4]で挙げられているTutorialの例の一部である．

```
globals[
    num-clusters
]

turtles-own
[
    time-since-last-found
]

to setup
    ca
    set num-clusters 4

    ask n-of num-clusters patches
    [
        ask n-of 20 patches in-radius 5
        [
            set pcolor red
        ]
    ]

    crt 2
    [
        set size 2
        set color yellow
        set time-since-last-found 999
        pen-down
    ]
    reset-ticks
end
```

Fig. 4: Mushroom のソースコード

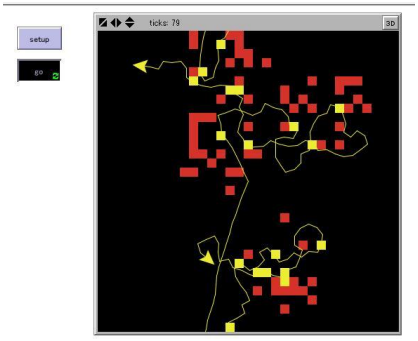

Fig. 5: Mushroom 実行図

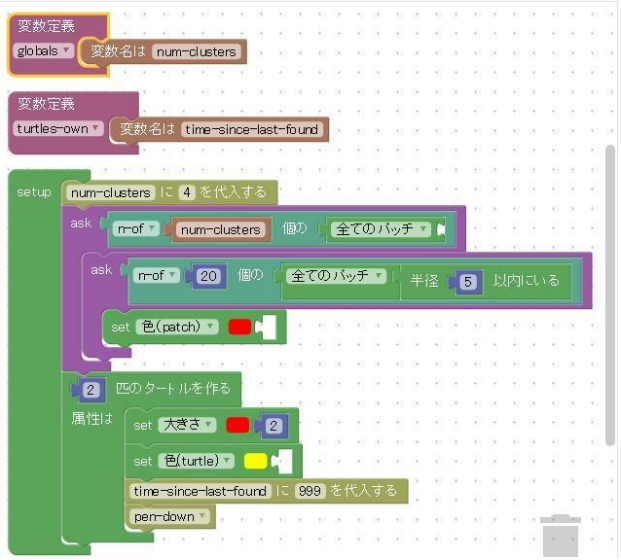

Fig. 6: ブロック言語での実装

4. SOARS におけるビジュアルブロックの検討

SOARSシミュレーションシステムで提供される言語のブロック化を検討し，その設計の検討を行った．

4.1 SOARS によるモデル化の概要および対応するブロックの構成

以下では，SOARS のモデル化の概略を説明しつつ，対応するブロック言語の構成を示す. SOARS におけるシミュレーションは，エージェント，スポット，ステージ，役割の 4 種類を用いて記述される．

エージェント，スポットは，それぞれ主体，主体の行動の場としてモデル化され，属性の集合として表現される．本プロトタイプでは，Fig.7のように属性のリストとして表現した．役割も属性の一つと考えられるが，ここでは別に「初期役割」というブロックを用意している．これらによりエージェントやスポットの大枠を定義する．

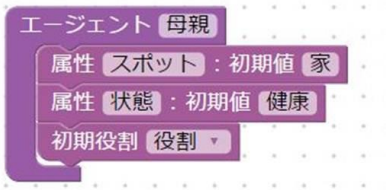

Fig. 7: エージェントの定義

ステージ：SOARS では 1 ステップ（ティック）の実行は，ステージと呼ばれる時間の論理概念の構成によって決定される．ブロック言語においては，単純にステージを線形に配置した(Fig.8).

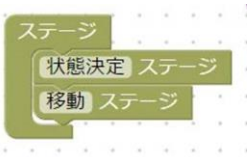

Fig. 8: ステージ構成を表すブロック

役割：エージェント，スポットは役割を持つことができる．役割は，それを保持するエージェントの挙動を特徴づける．各ステージにおける，挙動を(ステージ，条件，コマンド)の組で定義する．Fig.9 は，役割のブロックによる定義例を示す.

Fig. 9: 役割の定義例

以上の構成要素でシミュレーションは定義される．シミュレーションの実行は，定義されたステージ列は各ティックで同じ順番に呼び出され，それぞれのステージでは，そのステージでの行動ルールをもつ役割が発火され，エージェントやスポットが動作する．このような制御構造はシミュレーション言語に組み込まれているため，モデルの中に陽に記述する必要はない．

上記の実行モデルはステージ構成と役割に基づくシミュレーションの因果関係を記述しただけであるが，実際的なシミュレーションでは時間の単位やティックの粒度などを設定する必要がある．この機能は次のようなブロックによって実装した（Fig.10）．

Fig. 10: 時間設定の例

4.2 役割と振る舞いのブロック構成

役割をブロック言語で表す方法はいくつか考えられるが，本プロトタイプでは役割名とステージ名を表すブロックを先頭として，その下に続いて制御ブロックを配置するようにした．本プロトタイプでは（役割，ステージ）というペアに対して，行うべき「振る舞い」を対応させるというアプローチである．制御ブロックには，条件判定を行う条件ブロック，条件判定の結果によって実行されるコマンドブロックの 2 種類が接続される．

SOARS における条件判定やコマンドには, エージェントが意志決定とエージェントの行動を簡単に定義するための特徴的な命令が含まれている．本研究では, 代表的なものを実装した（Fig.12, Fig.13）．

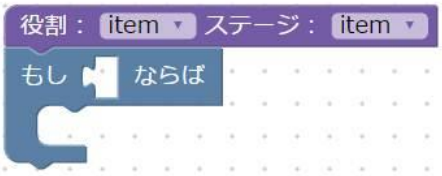

Fig. 11: 役割の定義法

Fig. 12: 条件ブロック

Fig. 13: コマンドブロック

5. 議論，考察

本研究では，二つのシミュレーション向け環境について，ビジュアルブロックによるシミュレーション記述の可能性を探った．作成した代表的なシミュレーションモデルから次のような知見が得られた．ビジュアルプログラミング言語の強みをシミュレーションの記述で活かせる．ブロック言語による記述の一番の特徴は，マウスなどによる直感的な操作でプログラムを構成できる点である．この点は，シミュレーション構築という文脈では次のような利点をもたらす．(1)命令などを構成する際に，ブロックを移動させ，接続したり切り離したりする操作で迅速な編集が可能である．これは編集(edit)，実行(run)，観察(observe)という開発のフィードバックループをまわすことに貢献する．さらに，シミュレーションの閲覧者が挙動について理解を深めるためにその構成を変更する，という処理が可能である．(2)テキスト表現に比べると，言語構成要素自体に表現力をもたせることができる．すなわち，自己説明的なブロックを与えることができれば，ブロックがそのプログラミング言語のリファレンスの役割を果たすため学習コスト軽減，開発効率向上につながる．

構成されたブロックの短所としては以下があげられる．(1)既存の方法のみでは，算術式のような表現は構成に手間がかかる上に可読性に欠ける．Fig.14 にあげた二つのブロックは表現方法が異なるが，どちらも算術式を表現している．

Fig.14 の上のブロックは 3 節で述べた型チェックを厳密に行ったブロックであり，下のブロックは式をテキストで自由記述できるようになっている．上のブロックでは取りうる値を型で限定しているため，バグを起こしにくいがブロックでの記述が冗長となり幅と記述にかかる時間が増す．一方，下のブロックはテキストで自由に書けるため，記述にかかる時間と幅をとらないが，ユーザーが取りうるべきでない値を書き込む可能性がある．シミュレーションにおいては，数式で挙動を表すことは重要であり，決定的である．解決策としてはテキストや数式を表現できる言語表現を併用することが考えられる．一方で，ブロック型ビジュアル言語の単純さを失わない手法が求められる．

(2)テキストに比べて視認性に欠ける場合がある．ブロックはテキストに比べ幅をとるので，ソースコード量が多くなるとブロックの数も増えすぐにエディタに収まりきらなくなる．そうなると，プログラムの全体像を把握できなくなりバグを引き起こすもとになる．

これからの発展としてはライブプログラミングを可能にすることが考えられる．ライブプログラミングとはソースコード編集がプログラム実行に与える影響を即座に表示することである．今回の場合なら，ブロックでプログラムを作成している時にソースコードだけでなく実行結果も同時に表示させる．こうすることにより，いま設置したブロックが何を表すブロックなのかより理解しやすくなり，使用者の学習性を向上させることができると考える．

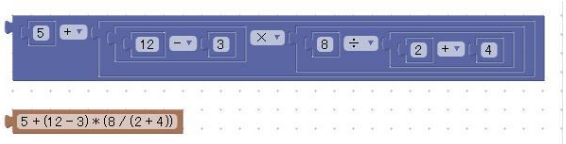

Fig. 14: 式を表す 2 種類のブロック

6. おわりに

本発表では，コミュニケーションツールとして効果的なエージェントシミュレーションシステムの実現に向けて，シミュレーション記述言語のビジュアルブロック化の試みを行った．この手法によって一定の効果が得られると考えられる．

参考文献

1) R. Mitchel et Al: Scratch: Programming for all, Communications of the ACM 52 (11): 60/67(2009)
2) U. Wilensky: NetLogo, https://ccl.northwestern.edu/netlogo/
3) 田沼英樹，出口弘，エージェントベース社会シミュレーション言語 SOARS の開発, 信学会論文誌，Vol.J90-D, No.9, 2415/2422 (2007)
4) S. F. Railsback and V. Grimm: Agent-based and Individual-based Modeling: A Practical Introduction, Princeton University Press(2011)